```
;*****************************************************************************
; Copyright (c) [01/26/1999] Scenix Semiconductor, Inc. All rights reserved.
;
 ;Scenix Semiconductor, Inc. assumes no responsibility or liability for
; the use of this [product, application, software, any of these products].
; Scenix Semiconductor conveys no license, implicitly or otherwise, under
; any intellectual property rights.
; Information contained in this publication regarding (e.g.: application,
; implementation) and the like is intended through suggestion only and may
; be superseded by updates. Scenix Semiconductor makes no representation
; or warranties with respect to the accuracy or use of these information,
; or infringement of patents arising from such use or otherwise.
;*****************************************************************************
; Filename: sx_modem_3_62.src
;
; Author:   Chris Fogelklou
;           Applications Engineer
;           Scenix Semiconductor Inc.
;
; Revision: 3.62
;
; Date:           January 26, 1999.
;
; Part:           SX28AC rev. 2.5
;
; Freq:           50Mhz
;
; Compiled using Parallax SX-Key software v1.0
;
; Version:  3.62
;
; Program Description:
;     This program is a full implementation of a half-duplex bell202 modem
;     which transmits and receives at 1200bps.  The baud rate of the uart
;     connected to it   should be set to 1200,N,8,1.
;     The necessary baud rate can be changed easily in this software by
;     using the defines present in the sx_demo software, which is available
;     on the web.  On power-up, this software generates a prompt with some
;     instructions on usage.  This version of   of the software has these
;     functions implemented...
;     - DTMF output for dialing
```

```
;       - FSK output
;       - FSK input
;       - 1200 baud UART for communication with a PC or terminal
;       - Small AT command set
;       - Ring detection
;       - Buffer for AT command storage
;       - String parser for AT command processing.  Allows easy addition of
;       - AT commands.
;
;         Authors: Chris Fogelklou, Scenix Semiconductor Inc.
;         Written: 98/12/18 to 98/12/21
;       Version: 3.62
;       Revisions: 3.0 began with the tx_modem_1_0.src software and combined it
;                  with the fsk_rx_1_0 software.
;                  3.1 modified so that UART speed = Transmit speed and this
;                  is settable by changing the divider_bit
;                  3.2 has much better documentation inside the software, and
;                  it also includes a transmit-only mode for error-free file
;                  transmission at 1200 baud and a receive-only mode for
;                  error-free file reception at 1200 baud.  Future revisions
;                  will fix the 1200-baud file transfer errors.  Debugging work
;                  is involved.
;                  3.4 fixes the bug in the DTMF generation code, which was
;                  causing the output signal to clip and it also adds "twist"
;                  to the DTMF generation code, which is required.  The high
;                  frequency signal should be 1.25 times greater in amplitude
;                  than the low frequency signal. (1/5/99)
;                  3.5 Fixes the bug in the UART code which is causing
;                  the modem to miss characters during file transfer.
;                  (UART process)
;                  3.51 Fixes a bug caused by the bug fix in 3.5, in which the
;                  pwm DC value is not reset to 2.5V when the output is disabled
;                  3.6      Adds an AT-command set and RING DETECT to the modem, for
;                  auto-answer in the AT command set.
;                  3.61 Adds ATO command to switch back to data mode and also
;                  adds additional documentation to the source code. (2/8/99)
;                  Removed the simplex modes of communication.
;                  3.62 Documentation updates.  A lot of old comments were left
;                  in from rev 3.51.  Updated ascii buffer to 63 bytes.  Ascii
;                  buffer space can be re-used for other variables, since its
;                  operations use the FSR only.  The actual RAM usage for the
```

```
;               buffer is limited to the number of characters stored + 1.
;               but it is capable of handling up to 63 bytes.(2/23/99)
;
;
;     I/O USAGE: INPUTS:
;
;     rx_pin            equ   ra.1  ; RS-232 input pin
;     fsk_in            equ   rb.1  ; FSK input pin
;
;               OUTPUTS:
;
;     PWM_pin           equ   ra.0  ; PWM output for D/A
;     tx_pin            equ   ra.2  ; RS-232 output pin
;     in_out            equ   ra.3  ; Enables/Disables output
;                             ; on SX DTMF DEMO boards.
;     led_pin           equ   rb.0  ; LED output pin
;     fsk input pin     is    rb.1  ; does not need a define, uses a bitmask
;     hook        equ   rb.4  ; Selects on-hook/off-hook
;
;     RESOURCES:
;     Program Memory:   TBD
;     Data Memory:      TBD
;*****************************************************************************
; Device Directives
;*****************************************************************************
            device      pins28,pages4,banks8          ; 28-pin device, 4 pages, 8 banks of RAM
            device      oschs,turbo,optionx,stackx    ; High speed oscillator, turbo mode,
                                                      ; option register extend, 8-level stack
            freq  50_000_000                          ; default run speed = 50MHz
            ID    'sx_mod36'                          ; Version = 3.62

            reset start                               ; JUMP to start label on reset
;*****************************************************************************
; Watches (For Debug in SX_Key software V.1.0 +)
;*****************************************************************************
            watch freq_acc_high,16,uhex
            watch freq_count_high,16,uhex
            watch freq_count_high2,16,uhex
            watch byte,1,fstr
            watch curr_sin,8,sdec
            watch sinvel,8,sdec
```

```
            watch pwm0,8,udec
            watch ring_duration,8,udec
            watch timer_flag,1,ubin
            watch timer_l,16,uhex
            watch temp,8,uhex
            watch fsk_bit_delay,8,uhex
            watch fsk_last_bit,1,ubin
            watch fsk_rx_en,1,ubin
            watch flags,8,ubin
            watch ascii_buffer,16,zstr
            watch ascii_buffer2,16,zstr
            watch ascii_buffer3,16,zstr
            watch ascii_buffer4,16,zstr
            watch ascii_index,8,udec
            watch fsr,8,udec
            watch indf,1,fstr
            watch wreg,1,fstr
            watch rings,8,udec
            watch plus_count,8,udec

;*************************************************************
; Macros
;*************************************************************
;     old_board               ; uncomment this if using with old boards.
;                             ; Boards marked REV1.4 are newer.  Boards
;                             ; Boards with no rev number are older.
;                             ; If board is green and uses a DIP package,
;                             ; keep the oldboard macro commented.

;*************************************************************
enable_o    macro 0           ; This macro enables the output
;*************************************************************

ifdef       old_board
            clrb  in_out            ; switch on the new modem boards.
  else
            setb  in_out
endif
            clr   flags
            endm
;*************************************************************
```

```
;*************************************************************
; Equates for the FSK receive part of the modem
;*************************************************************
glitch_th         equ   10    ; The threshold which defines a glitch (small spike which should be
ignored)
low_count_error_th      equ   30    ; The lowest count allowed for a high frequency
low_high_th       equ   95    ; The lowest count allowed for a low frequency
high_count_error_th     equ   150   ; The highest count allowed for a low frequency

; *** 1200 baud using a 1/2 counter.
baud_bit    =       7                       ;for 1200 baud fsk
start_delay =       128+64+1                ; "    "    "
divider_bit =     1                 ;1 for 1200 baud, 2 for 600 baud, 3 for 300 baud.


;*************************************************************************
; Equates for common data comm frequencies (DTMF generation)
;*************************************************************************
f697_h            equ   $012  ; DTMF Frequency
f697_l            equ   $09d

f770_h            equ   $014  ; DTMF Frequency
f770_l            equ   $090

f852_h            equ   $016  ; DTMF Frequency
f852_l            equ   $0c0

f941_h            equ   $019  ; DTMF Frequency
f941_l            equ   $021

f1209_h           equ   $020  ; DTMF Frequency
f1209_l           equ   $049

f1336_h           equ   $023  ; DTMF Frequency
f1336_l           equ   $0ad

f1477_h           equ   $027  ; DTMF Frequency
f1477_l           equ   $071

f1633_h           equ   $02b  ; DTMF Frequency
```

```
f1633_l             equ   $09c

f1300_h             equ   $022  ; 1300Hz Signifies HIGH data in Bell202 Spec
f1300_l             equ   $0b7

f2100_h             equ   $038  ; 2100Hz Signifies LOW data in Bell202 Spec
f2100_l             equ   $015


;*****************************************************************************
; Pin Definitions
;*****************************************************************************
PWM_pin             equ   ra.0        ; PWM output for D/A
rx_pin              equ   ra.1        ; RS-232 Input pin
tx_pin              equ   ra.2        ; RS-232 Output pin
in_out              equ   ra.3        ; Switches between output
                                ; and input on SX DTMF DEMO boards.
led_pin             equ   rb.0        ; Flashes while characters are
                                ; being received.
;fsk input pin      is    rb.1        ; does not need a define, uses a bitmask
ring        equ     rb.3        ; Ring detection pin
hook        equ     rb.4        ; Goes on/off-hook.
;*****************************************************************************

;*****************************************************************************
; Global Variables
;*****************************************************************************
            org   $8          ; Global registers

flags               ds    1
      rx_flag             equ   flags.0     ; Signifies a bit recieved via. RS-232
      dtmf_gen_en equ   flags.1     ; Signifies whether or not DTMF output is enabled
      fsk_tx_en   equ   flags.2 ; These flags are the same and they both
      fsk_transmitting equ    flags.3 ; indicate when the UART is transmitting
      timer_flag  equ   flags.4     ; Flags a rollover of the timers.
      fsk_rx_en   equ   flags.5 ; Enables the FSK receiver.
      fsk_rx_flag equ   flags.6 ; Signifies reception of FSK
      ring_det_en equ   flags.7 ; Enables the ring detector

temp                ds    1     ; Temporary storage register for use by the main program
divider                   ds    1     ; used to divide down the UART to 1200 baud
```

```
IRQ_temp          ds    1     ; Temporary register for use by the interrupt service routine
ascii_index       ds    1     ; Register used for the ascii buffering
command_index           ds    1     ; Register used as an index to the command to compare to.


;*************************************************************
; Bank 0 Variables
;*************************************************************
            org   $10

sin_gen_bank      =     $

freq_acc_high           ds    1     ;
freq_acc_low            ds    1     ; 16-bit accumulator which decides when to increment the sine wave

freq_acc_high2          ds    1     ;
freq_acc_low2           ds    1     ; 16-bit accumulator which decides when to increment the sine wave

freq_count_high         ds    1     ; freq_count = Frequency * 6.83671552
freq_count_low          ds    1     ; 16-bit counter which decides which frequency for the sine wave

freq_count_high2  ds    1     ; freq_count = Frequency * 6.83671552
freq_count_low2         ds    1     ; 16-bit counter which decides which frequency for the sine wave

curr_sin          ds    1     ; The current value of the imitation sin wave
sinvel                  ds    1     ; The velocity of the sin wave

curr_sin2         ds    1     ; The current value of the imitation sin wave
sinvel2                 ds    1     ; The velocity of the sin wave

PWM_bank    =     $

pwm0_acc          ds    1           ; PWM accumulator
pwm0              ds    1           ; current PWM output

timers            =     $

timer_l                 ds    1
timer_h                 ds    1

;*************************************************************
```

```
; Bank 1 Variables
;*************************************************************
            org     $30                     ;bank3 variables

serial              =       $                       ;UART bank

tx_high             ds      1                       ;hi byte to transmit
tx_low              ds      1                       ;low byte to transmit
tx_count    ds      1                       ;number of bits sent
tx_divide   ds      1                       ;xmit timing (/16) counter
rx_count    ds      1                       ;number of bits received
rx_divide   ds      1                       ;receive timing counter
rx_byte             ds      1                       ;buffer for incoming byte
string              ds    1
byte        ds    1
plus_count  ds    1                 ;counts the number of '+'s received
                                    ;while in FSK_IO mode.

;*************************************************************
; Bank 2 Variables
;*************************************************************
            org   $50

fsk_transmit_bank =     $

fsk_bit_delay     ds    1
fsk_tx_byte ds    1
fsk_flags   ds    1
      fsk_last_bit      equ   fsk_flags.0

fsk_tx_counter    ds    1

fsk_receive_bank  =     $

      fsk_trans_count   ds    1           ; This register counts the number of counts
                                    ; between transitions at the pin
      rb_past_state     ds    1           ; This register keeps track of the previous

fsk_rx_count      ds      1                       ; number of bits received
fsk_rx_divide     ds    1                 ; bit delay
fsk_rx_byte ds      1                     ; buffer for incoming byte
```

```
      fsk_current_in    equ   fsk_flags.1 ; The bit represented by the current input frequency
      fsk_trans   equ   fsk_flags.2

;*************************************************************
; Bank 3 Variables
;*************************************************************
            org   $70

      ring_detect_bank  =     $

      ring_timer_low          ds    1
      ring_timer_high         ds    1


      ring_flags        ds    1
      ring_timer_flag         equ   ring_flags.0
      ringing_1         equ   ring_flags.1
      ringing_2         equ   ring_flags.2
      long_ring         equ   ring_flags.3
      ring_pause        equ   ring_flags.4
      ring_captured           equ   ring_flags.5
      ring_off_timer_1  ds    1
      ring_timer        ds    1
      ring_duration           ds    1
      ring_duration_timer     ds    1
      answer_rings            ds    1
      rings             ds    1
;*************************************************************
; Bank 4, 5, 6, 7 (for ascii buffer, but can be reused.)
;*************************************************************
            org   $90
      ascii_buffer            =     $
            org   $b0
      ascii_buffer2           =     $
            org   $d0
      ascii_buffer3           =     $
            org   $f0
      ascii_buffer4           =     $

;*************************************************************************
; Interrupt
```

```
;
; With a retiw value of -163 and an oscillator frequency of 50MHz, this
; code runs every 3.26us.
;*****************************************************************************
            org   $0                ; The interrupt Service routine starts at location zero.
;*****************************************************************************
PWM_OUTPUT
; This outputs the current value of pwm0 to the PWM_pin.  This generates
; an analog voltage at PWM_pin after filtering.
; INPUTS:
;     pwm0 -  The value from 0-255 representing the analog voltage to be
;             output by the PWM_pin
;*****************************************************************************
            bank  PWM_bank
            add   pwm0_acc,pwm0                 ; add the PWM output to the accumulator
            snc
            jmp   :carry                        ; if there was no carry, then clear
                                              ; the PWM_pin
            clrb  PWM_pin
            jmp   PWM_out
:carry
            setb  PWM_pin                       ; otherwise set the PWM_pin
PWM_out
;*****************************************************************************
TASK_SWITCHER
; Now decide which task to do... we may only do one, and the transmit
; takes priority over the receive functions.
; INPUTS:
;     flags - Depending on which of the flags are set in the flags register,
;             either FSK transmission, FSK reception, or DTMF generation
;             will occur.
;*****************************************************************************

            snb   ring_det_en
            call  @ring_detect

            jnb   fsk_tx_en,:fsk_tx_out         ; If FSK transmit is enabled
            call  @sine_generator2            ; only use one of the sine generators
            call  @FSK_TX_UART                  ; perform the transmit UART first
            jmp   :TASK_OUT                   ; and then skip over DTMF because can't do
:fsk_tx_out                                   ; both at once
```

```
            jnb   dtmf_gen_en,:sine_gen_out     ; if dtmf generation is enabled
            call  @sine_generator1        ; do it.
            jmp   :TASK_OUT
:sine_gen_out
            jnb   fsk_rx_en,:fsk_rx_out         ; jump out if the FSK receiver is not enabled
            call  @FSK_RECEIVE
            jmp   :TASK_OUT
:fsk_rx_out
:TASK_OUT
;*************************************************************************
:transmit
; This is an asynchronous transmitter for RS-232 transmission
; INPUTS:
;     divider.divider_bit -   Transmitter/receiver only executes when this bit is = 1
;     tx_divide.baud_bit  -   Transmitter only executes when this bit is = 1
;     tx_high               - Part of the data to be transmitted
;     tx_low                - Some more of the data to be transmitted
;     tx_count        - Counter which counts the number of bits transmitted.
; OUTPUTS:
;     tx_pin                - Sets/Clears this pin to accomplish the transmission.
;*************************************************************************
            jnb   divider.divider_bit,:rxdone   ; cut the UART speed down to 1200/600/300
                                          ; depending on divider bit
            bank  serial
            clrb    tx_divide.baud_bit      ;clear xmit timing count flag
            inc     tx_divide               ;only execute the transmit routine
            STZ                             ;set zero flag for test
            SNB     tx_divide.baud_bit      ; every 2^baud_bit interrupt
            test    tx_count                ;are we sending?
            JZ      :receive                ;if not, go to :receive
            clc                             ;yes, ready stop bit
            rr      tx_high                 ; and shift to next bit
            rr      tx_low                  ;
            dec     tx_count                ;decrement bit counter
            movb    tx_pin,/tx_low.6        ;output next bit

;*************************************************************************
:receive
; This is an asynchronous receiver for RS-232 reception
; INPUTS:
;     rx_pin              -  Pin which RS-232 is received on.
```

```
; OUTPUTS:
;     rx_byte              -  The byte received
;     rx_flag              -  Set when a bRyte is received.
;*************************************************************************
            movb    c,rx_pin                ;get current rx bit
            test    rx_count                ;currently receiving byte?
            jnz     :rxbit                  ;if so, jump ahead
            mov     w,#9                    ;in case start, ready 9 bits
            sc                              ;skip ahead if not start bit
            mov     rx_count,w              ;it is, so renew bit count
            mov     rx_divide,#start_delay  ;ready 1.5 bit periods
:rxbit            djnz    rx_divide,:rxdone       ;middle of next bit?
            setb    rx_divide.baud_bit      ;yes, ready 1 bit period
            dec     rx_count                ;last bit?
            sz                              ;if not
            rr      rx_byte                 ;  then save bit
            snz                             ;if so
            setb    rx_flag                 ;  then set flag
:rxdone

;*************************************************************************
do_timers
; The timer will tick at the interrupt rate (3.26us for 50MHz.)  To set up
; the timers, move in FFFFh - (value that corresponds to the time.)  Example:
; for 1ms = 1ms/3.26us = 306 dec = 132 hex so move in $FFFF - $0132 = $FECD
;*************************************************************************

            bank  timers                    ; Switch to the timer bank
            mov   w,#1
            add   timer_l,w         ; add 1 to timer_l
            jnc   :timer_out        ; if it's not zero, then
            add   timer_h,w         ; don't increment timer_h
            snc
            setb  timer_flag
            movb  led_pin,timer_h.6 ; once timer_h is changed, update the LED
:timer_out
            clrb  divider.divider_bit
            inc   divider                   ; do nothing unless divider_bit is a '1'

;*************************************************************************
:ISR_DONE
```

```
; This is the end of the interrupt service routine.  Now load 163 into w and
; perform a retiw to interrupt 163 cycles from the start of this one.
; (3.26us@50MHz)
;*******************************************************************************
            mov   w,#-163           ;1    ; interrupt 163 cycles after this interrupt
            retiw             ;3    ; return from the interrupt
;*******************************************************************************
; End of the Interrupt Service Routine
;*******************************************************************************
;*******************************************************************************
start
; Program Starts Here on Power Up
;*******************************************************************************
            jmp   start_2
_FSK_IO           jmp   FSK_IO                  ; Jump table for FSK_IO


start_2           call  @zero_ram         ; clear all ram contents
            call  @init             ; initialize all important registers.

      ;*****************************************************************************
      ; Main Code:
      ; -Sends Prompt
      ; -Waits for input from UART
      ; -Pushes incoming characters onto the buffer.
      ;    -If incoming character is a backspace, deletes last character from
      ;     buffer
      ; -On CR character, jumps to the parse_string routine
      ; -Outputs RING if a ring is detected
      ; -Answers if the modem is set for auto-answer mode and a ring is detected,
      ;  after number of rings specified occurs.
      ;*****************************************************************************
            mov   w,#_hello               ; send hello string
            call  @send_string

            mov   w,#_instructions        ; send instructions
            call  @send_string

_send_prompt      mov   w,#_CR
            call  @send_string
            mov   w,#_COMMAND_MODE
```

```
            call  @send_string
            mov   w,#_prompt              ; send prompt
            call  @send_string
            clr   flags

_cmd_loop   jb    hook,:not_auto_answer              ; If we are off-hook, don't
            setb  ring_det_en                 ; watch for ring.
            bank  ring_detect_bank
            jnb   ring_captured,:not_auto_answer
            clrb  ring_captured
            cjb   ring_duration,#27,:not_auto_answer  ; If the ring was less than
            inc   rings                       ; 200ms, ignore it, otherwise
            mov   w,#_RING                    ; increment the ring count.
            call  @send_string                     ; and send 'RING' to the terminal
            bank  ring_detect_bank
            mov   w,answer_rings                   ; If answer_rings is 0, then this
            jz    :not_auto_answer            ; is not auto answer.
            xor   w,rings                          ; Else, if # rings = answer_rings, answer
            jz    :answer

:not_auto_answer
            jnb   rx_flag,_cmd_loop       ; wait for an input character
            clrb  rx_flag                      ; from terminal
            bank  serial
            mov   byte,rx_byte

            call  @uppercase              ; convert it to uppercase
            cje   byte,#$20,_cmd_loop          ; if it equals a space, ignore it.
            cje   byte,#$0d,:enter        ; if it equals a carriage return, parse the string.
            mov   w,byte                       ; if it does not resemble the above characters, echo
it.
            call  @send_byte
            cje   byte,#$08,:backspace         ; if it equals a backspace, delete one character in
the buffer.
            call  @buffer_push                 ; otherwise, store it
            jmp   _cmd_loop               ; and come back for more.

:answer
            setb  hook                    ; answer the phone!!!
            clrb  ring_det_en             ; disable ring detect.
            mov   w,#_ANSWERING                ; send 'answering' to the terminal
```

```
            call  @send_string
            call  _FSK_IO                       ; go to half-duplex mode.
            jmp   _cmd_loop                 ; and go back for more.

:backspace
            call  @buffer_backspace
            jmp   _cmd_loop

:enter                                          ; If the user presses enter, then parse the string.

;*************************************************************************
; String parser (Checks to see if buffer = any commands)
; -Checks contents of ascii buffer against any commands stored in ROM
; -If a command = the contents of the ascii buffer, a routine will be called
; -Each routine MUST perform a retw 0 on exit, or parse_string will not
;  know that a routine has run and it should exit back to command mode.
; -Exits back to command mode when it detects a zero after the table look-up.
; -Outputs 'OK' if no commands are matched.
;*************************************************************************
parse_string
            clr   ascii_index               ; Clear the index into the ascii buffer
            clr   command_index                 ; And the index into the commands
:loop       call  @buffer_get               ; Get a vale from the buffer at ascii_index
            call  command_table                 ; Get a character from one of the commands
            test  w                         ; If the return value is 0, then this matched
            jz    :done                     ; the command and ran a routine.  Exit.
            bank  serial
            xor   w,byte                        ; compare the command's character with the
            jnz   :not_equal                ; buffer's character.
            call  @inc_ascii_index          ; Increment the index into the buffer.
            jmp   :loop
:not_equal
            inc   command_index                 ; If the buffer did not equal the command,
            clr   ascii_index               ; start from the beginning of a new command
            cjne  command_index,#6,:loop        ; and the buffer.  (This number = # of commands)
            mov   w,#_OK                        ; If we have checked all 4 commands, then this
            call  @send_string                  ; did not equal any so send an 'OK' message.
:done
            bank  ascii_buffer
            clr   ascii_index
            clr   ascii_buffer
```

```
            jmp   _send_prompt

;*************************************************************************
command_table
            mov   w,command_index
            add   pc,w
            jmp   command_1
            jmp   command_2
            jmp   command_3
            jmp   command_4
            jmp   command_5
            jmp   command_6
;*************************************************************************
command_1                           ; Dial command
            mov   w,ascii_index
            add   PC,w
            retw  'A'
            retw  'T'
            retw  'D'
            retw  'T'
            jmp   DIAL_MODE
;*************************************************************************
command_2                           ; Hang up command
            mov   w,ascii_index
            add   PC,w
            retw  'A'
            retw  'T'
            retw  'H'
            jmp   HANG_UP
;*************************************************************************
command_3                           ; Initialize
            mov   w,ascii_index
            add   PC,w
            retw  'A'
            retw  'T'
            retw  'Z'
            jmp   INITIALIZE
;*************************************************************************
command_4                           ; Answer/ Auto answer
            mov   w,ascii_index
            add   PC,w
```

```
            retw  'A'
            retw  'T'
            retw  'A'
            jmp   AUTO_ANSWER
;*************************************************************************
command_5                           ; Data mode
            mov   w,ascii_index
            add   PC,w
            retw  'A'
            retw  'T'
            retw  'O'
            jmp   FSK_IO
;*************************************************************************
command_6                           ; Help
            mov   w,ascii_index
            add   PC,w
            retw  '?'
            jmp   HELP
;*************************************************************************
; END of String parser (Checks to see if buffer = any commands)
;*************************************************************************


      ;*************************************************************************
      ; Dial Mode:
      ; -Dials contents of ascii buffer, starting from location pointed
      ;  to by ascii_index.
      ; -Responds to these commands:
      ;     0-9, *, #    - Dials the specified number
      ;     ,            - Pause for 2 seconds
      ; -Jumps to data mode after dialing.
      ;*************************************************************************
DIAL_MODE
            clrb  ring_det_en
            mov   w,#_CR
            call  @send_string
            mov   w,#_DIALING             ; send Dialing
            call  @send_string
            setb  hook                    ; pick up the line

:dial_loop  call  @buffer_get             ; wait for an input character
```

```
            call  @uppercase              ; convert it to uppercase
            mov   w,byte
            snz
            jmp   FSK_IO
            call  @send_byte
            cje   byte,#',',:pause        ; if the character = ',', pause for 2s
            call  @digit_2_index               ; convert the ascii digit to an
                                          ; index value
            call  @load_frequencies       ; load the frequency registers
            call  @dial_it                ; dial the number for 60ms and return.
:inc        call  @inc_ascii_index        ; increment the index into the table
            jmp   :dial_loop
:pause
            mov   w,#201                       ; delay 2s
            call  @delay_10n_ms
            jmp   :inc
      ;*************************************************************************
      ; FSK Input/Output mode:
      ; -Sends Prompt
      ; -Sends any characters received from the terminal through the phone line
      ;  and sends any characters received from the phone line to the terminal
      ; -performs retw 0 when it detects incoming '+++' from UART
      ;*************************************************************************
FSK_IO

            mov   w,#_CR
            call  @send_string
            mov   w,#_DATA_MODE                ; send FSK I/O string
            call  @send_string
            mov   w,#_PROMPT
            call  @send_string

            bank  sin_gen_bank
            mov   curr_sin2,#-4                ; set up so the wave starts at close to the right spot
(0).
            mov   sinvel2,#-8
            clr   flags
            setb  fsk_rx_en               ; enable the FSK detector
            bank  serial
:RESET_PLUSES     mov   plus_count,#3               ; counts the number of '+'s received
```

```
:TX_LOOP
            jb    fsk_rx_flag,:fsk_byte_received      ; if FSK received a byte, go to
:fsk_byte_received
            jb    rx_flag,:byte_received        ; if the UART received a byte, go to :byte_received
            jmp   :TX_LOOP

:byte_received
            bank  serial                        ; echo the character back to the terminal
            mov   byte,rx_byte
            clrb  rx_flag                       ; clear the flag
            call  @fsk_transmit_byte            ; send the byte via. FSK
            bank  serial
            mov   w,byte
            xor   w,#'+'
            jnz   :RESET_PLUSES
            dec   plus_count
            snz
            retw  0
            jmp   :TX_LOOP                ; return to the loop

:fsk_byte_received
            bank  fsk_receive_bank        ; send the character to the terminal
            mov   w,fsk_rx_byte
            clrb  fsk_rx_flag             ; clear the flag
            call  @send_byte
            jmp   :TX_LOOP                ; return to the loop

;*****************************************************************************
HANG_UP           ; goes on-hook
            mov   w,#_CR
            call  @send_string
            mov   w,#_OK
            call  @send_string
            clrb  hook
            retw  0

;*****************************************************************************
INITIALIZE  ; calls init routine
            mov   w,#_CR
            call  @send_string
            mov   w,#_OK
```

```
            call  @send_string
            call  @init
            retw  0
;*************************************************************************
HELP        ; outputs help stuff
            mov   w,#_plus
            call  @send_string
            mov   w,#_COMMAND_MODE
            call  @send_string
            mov   w,#_CR
            call  @send_string
            mov   w,#_ATO
            call  @send_string
            mov   w,#_DATA_MODE
            call  @send_string
            mov   w,#_ATDT
            call  @send_string
            mov   w,#_DIALING
            call  @send_string
            mov   w,#_CR
            call  @send_string
            mov   w,#_ATA
            call  @send_string
            mov   w,#_AUTO_ANSWER
            call  @send_string
            mov   w,#_ATH
            call  @send_string
            mov   w,#_HANGING_UP
            call  @send_string
            mov   w,#_ATZ
            call  @send_string
            mov   w,#_INIT
            call  @send_string
            retw  0
;*************************************************************************
AUTO_ANSWER ; Moves a default value of 1 into answer_rings.  This
            ; specifies the number of rings to answer after.  If the
            ; user has entered ATA3, for instance, then a 3 will
            ; replace the 1 in answer_rings.  The modem will answer
            ; after 3 rings.  If the user enters ATA, the modem will
            ; answer after 1 ring.  ATA0 should force the modem to
```

```
            ; pick up immediately.
            bank  ring_detect_bank
            clr   rings
            mov   answer_rings,#1
            call  @buffer_get       ; get the next character from
                                    ; the ascii_buffer.
            bank  serial
            mov   w,byte                ; If this character is not a null,
            jz    :done             ; then use it to indicate how many
            sub   byte,#'0'         ; rings to trigger on.
            mov   w,byte
            bank  ring_detect_bank
            mov   answer_rings,w

:done       mov   w,#_CR
            call  @send_string
            mov   w,#_AUTO_ANSWER
            call  @send_string
            retw  0


org   $200
;*************************************************************************
;     Miscellaneous subroutines
;*************************************************************************
;*************************************************************************
; Initialization Code
;*************************************************************************
init
            call  @zero_ram
            bank  sin_gen_bank
;           mov   curr_sin,#32              ;init variables.  A sine starts at 1, a cos wave
starts at 0.
;           mov   sinvel,#0
            mov   curr_sin,#-4              ; use these values for a wave which is 90 degrees out
of phase.
            mov   sinvel,#-8
            mov   curr_sin2,#-4             ; use these values for a wave which is 90 degrees out
of phase.
            mov   sinvel2,#-8
            call  @disable_o
```

```
            mov   m,#$0c
            mov   !rb,#%11111101                ; enable Schmidt trigger on rb1 (for FSK receive)
            mov   m,#$D                   ; make ra0 cmos-level
            mov   !ra,#%1110
            mov   m,#$F

            mov   rb,#%11101110                 ; on-hook,led off.
            mov   !ra,#%0010              ; ra0 = PWM output, ra1 = rx_pin, ra2 = tx_pin, ra3 = in_out
            mov   !rb,#%00101110                ; rb4 = hook, rb0 = led pin, rb7,6 is used for
debugging

            mov   !option,#%00011111            ; enable wreg and rtcc interrupt

retp
;*****************************************************************************
buffer_push
; This subroutine pushes the contents of byte onto the 32-byte ascii buffer.
;*****************************************************************************
      bank  serial                  ; Move the byte into the buffer
      mov   temp,byte
      mov   fsr,#ascii_buffer
      add   fsr,ascii_index
      mov   indf,temp
                              ; Increment index and keep it in range
      call  @inc_ascii_index
      mov   fsr,#ascii_buffer ; Null terminate the buffer.
      add   fsr,ascii_index
      clr   indf
      bank  serial
      retp
;*****************************************************************************
;*****************************************************************************
buffer_backspace
; This subroutine deletes one value of the buffer and decrements the index
;*****************************************************************************
      dec   ascii_index
      and   ascii_index,#%01101111

      mov   fsr,#ascii_buffer
      add   fsr,ascii_index
```

```
      clr   indf
      bank  serial
      retp
;*************************************************************************
inc_ascii_index
; This subroutine increments the index into the buffer
;*************************************************************************
      mov   w,ascii_index
      and   w,#%00001111
      xor   w,#%00001111
      jnz   :not_on_verge
      inc   ascii_index
      mov   w,#16
      add   w,ascii_index
      and   w,#$7f
      mov   ascii_index,w
      retp
:not_on_verge
      inc   ascii_index
      retp
;*************************************************************************
buffer_get
; This subroutine retrieves the buffered value at index
;*************************************************************************
      mov   fsr,#ascii_buffer
      add   fsr,ascii_index
      mov   w,indf
      bank  serial
      mov   byte,w

      retp
;*************************************************************************
;*************************************************************************
delay_10n_ms
; This subroutine delays 'w'*10 milliseconds.
; This subroutine uses the TEMP register
; INPUT           w     -     # of milliseconds to delay for.
; OUTPUT    Returns after 10 * n milliseconds.
;*************************************************************************
      mov   temp,w
      bank  timers
```

```
:loop clrb  timer_flag  ; This loop delays for 10ms
      mov   timer_h,#$0f4
      mov   timer_l,#$004
      jnb   timer_flag,$
      dec   temp        ; do it w-1 times.
      jnz   :loop
      clrb  timer_flag
      retp
;*************************************************************************
delay_n_ms
; This subroutine delays 'w' milliseconds.
; This subroutine uses the TEMP register
; INPUT           w     -     # of milliseconds to delay for.
; OUTPUT    Returns after n milliseconds.
;*************************************************************************
      mov   temp,w
      bank  timers
:loop clrb  timer_flag  ; This loop delays for 1ms
      mov   timer_h,#$0fe
      mov   timer_l,#$0cd
      jnb   timer_flag,$
      dec   temp        ; do it w-1 times.
      jnz   :loop
      clrb  timer_flag
      retp
;*************************************************************************
zero_ram
; Subroutine - Zero all ram.
; INPUTS:   None
; OUTPUTS:  All ram locations (except special function registers) are = 0
;*************************************************************************
            CLR   FSR
:loop       SB      FSR.4                   ;are we on low half of bank?
            SETB    FSR.3                   ;If so, don't touch regs 0-7
            CLR     IND                     ;clear using indirect addressing
            IJNZ    FSR,:loop         ;repeat until done
            retp
;*************************************************************************
; Subroutine - Get byte via serial port and echo it back to the serial port
; INPUTS:
;     -NONE
```

```
; OUTPUTS:
;     -received byte in rx_byte
;*************************************************************************
get_byte          jnb     rx_flag,$       ;wait till byte is received
            clrb    rx_flag         ;reset the receive flag
            bank  serial
            mov     byte,rx_byte          ;store byte (copy using W)
                                    ; & fall through to echo char back
            retp
;*************************************************************************
; Subroutine - Get byte via Bell202 FSK and send it to the serial port
; INPUTS:
;     -NONE
; OUTPUTS:
;     -received byte in fsk_rx_byte
;*************************************************************************
fsk_get_byte      jnb     fsk_rx_flag,$         ;wait till byte is received
            clrb    fsk_rx_flag           ;reset the receive flag
            bank  fsk_receive_bank
            mov     byte,fsk_rx_byte      ;store byte (copy using W)
                                    ; & fall through to echo char back
;*************************************************************************
; Subroutine - Send byte via serial port
; INPUTS:
;     w     -     The byte to be sent via RS-232
;*************************************************************************
send_byte         bank    serial

:wait             test    tx_count                ;wait for not busy
            jnz     :wait                   ;

            not     w                       ;ready bits (inverse logic)
            mov     tx_high,w               ; store data byte
            setb    tx_low.7                ; set up start bit
            mov     tx_count,#10            ;1 start + 8 data + 1 stop bit
            RETP                            ;leave and fix page bits

;*************************************************************************
; Subroutine - Send string pointed to by address in W register
; INPUTS:
;     w     -     The address of a null-terminated string in program
```

```
;                memory
; OUTPUTS:
;     outputs the string via. RS-232
;*****************************************************************************
send_string bank  serial
            mov     string,w                ;store string address
:loop               mov     w,string                ;read next string character
            mov     m,#4                    ; with indirect addressing
            iread                           ; using the mode register
            mov     m,#$F                   ;reset the mode register
            test    w                       ;are we at the last char?
            snz                             ;if not=0, skip ahead
            RETP                            ;yes, leave & fix page bits
            call    send_byte               ;not 0, so send character
            inc     string                  ;point to next character
            jmp     :loop                   ;loop until done

;*****************************************************************************
; Subroutine - Make byte uppercase
; INPUTS:
;     byte  -     The byte to be converted
;*****************************************************************************
uppercase           csae    byte,#'a'             ;if byte is lowercase, then skip ahead
            RETP

            sub     byte,#'a'-'A'           ;change byte to uppercase
            RETP                            ;leave and fix page bits
;*****************************************************************************
; Subroutine - Disable the output (Enable the input)
;*****************************************************************************
disable_o
ifdef       old_board
            setb  in_out            ; set the analogue switch for
  else
            clrb  in_out
endif
            bank  PWM_bank    ; input mode.
            mov   pwm0,#128   ; put 2.5V DC on PWM output pin
            retp
;*****************************************************************************
org   $400
```

```
;*****************************************************************************
; Jump table for page 2
;*****************************************************************************
FSK_RECEIVE jmp   _FSK_RECEIVE
;*****************************************************************************
; String data (for RS-232 output) and tables
;*****************************************************************************
_hello          dw      13,10,'SX Modem V 3.6',13,10,0
_instructions     dw    '- ? For Help',0
_DIALING    dw    'DIAL ',0
_ANSWERING  dw    'ANSWERING ',0
_AUTO_ANSWER      dw    'AUTO ANSWER ',13,10,0
_RING       dw    'RING',13,10,0
_PROMPT           dw    13,10,'>',0
_HANGING_UP dw    'HANG UP ',13,10,0
_ATDT       dw    13,10,'ATDT=',0
_ATA        dw    'ATA =',0
_ATH        dw    'ATH =',0
_ATZ        dw    'ATZ =',0
_ATO        dw    'ATO =',0
_plus       dw    13,10,'+++ =',0
_OK         dw    'OK',13,10,0
_CR         dw    13,10,0
_COMMAND_MODE     dw    'COMMAND MODE',0
_DATA_MODE  dw    'DATA MODE',0
_INIT       dw    'INIT',0
;*****************************************************************************
; FSK transmit/receive functions
;*****************************************************************************
ring_detect jmp   _ring_detect
;*****************************************************************************
FSK_TX_UART ;(part of interrupt service routine)
; This subroutine creates an internal transmit UART using the data in
; fsk_tx_byte
;*****************************************************************************
            sb    fsk_transmitting
            RETP
            sb    divider.divider_bit                ; divide the baud by divider_bit
            RETP
            bank  fsk_transmit_bank
            clrb  fsk_bit_delay.7                    ; multiply the baud by 2
```

```
            dec   fsk_bit_delay                       ; Decrement the delay counter
            sz
            RETP
            stc                                 ; set the carry bit to create a stop bit
            rr    fsk_tx_byte
            sc
            clrb  fsk_last_bit
            snc
            setb  fsk_last_bit
            jb    fsk_last_bit,:new_bit_is_high
:new_bit_is_low
            bank  sin_gen_bank
            mov   freq_count_high2,#f2100_h           ; output a frequency of 2100Hz
            mov   freq_count_low2,#f2100_l
            jmp   :end_new_bit
:new_bit_is_high
            bank  sin_gen_bank
            mov   freq_count_high2,#f1300_h           ; output a frequency of 1300Hz
            mov   freq_count_low2,#f1300_l
:end_new_bit
            bank  fsk_transmit_bank
            decsz fsk_tx_counter
            RETP

:FSK_DONE_TRANSMITTING
            setb  fsk_last_bit                        ; since we're done transmitting,
            clrb  fsk_transmitting              ; clear the transmitting flag

            RETP
;*************************************************************************
fsk_transmit_byte
; This subroutine initializes the FSK UART and the sine generator and then
; it transmits data via FSK modulation to an outside source.  The byte to
; send is passed in the 'w' register.  Returns when byte transmission is
; done and FSK wave is close to zero OR when another character is received
; via. RS-232.  If another character is received via. RS-232, it immediately
; exits so that next character can begin transmission without re-initializing.
;*************************************************************************

            jb    fsk_transmitting,$      ; wait until done transmitting.
            bank  fsk_transmit_bank
```

```
            mov   fsk_tx_byte,w

:enable             enable_o                ; enable the outputs

            bank  fsk_transmit_bank
            clr   fsk_bit_delay             ; since fsk_bit_delay goes from 0 (256) to zero,
                                  ; clear fsk_bit_delay to set up for the first bit.
            clrb  fsk_last_bit              ; set fsk_last_bit to be an internal low (start bit)
            mov   fsk_tx_counter,#10        ; set up the bit counter for 1 start, 8 data, and 1 stop
            bank  sin_gen_bank
            mov   freq_count_high2,#f2100_h ; set up the sine generator to
            mov   freq_count_low2,#f2100_l ; output 2100 Hz.
            mov   curr_sin,#0       ; make the output of sin gen 1=0
                                    ; so it doesn't interfere with sin gen 2

            bank  fsk_transmit_bank ; enable the 2nd sin generator and the TX UART.
            setb  fsk_transmitting
            setb  fsk_tx_en

            bank  sin_gen_bank
:wait_loop  snb   rx_flag                   ; if another character is received, don't disable
            retp                    ; output
            jb    fsk_transmitting,:wait_loop ; otherwise, wait until we are done transmitting
            cjne  curr_sin2,#-4,:wait_loop ; and wait until FSK signal is relatively close to zero.
            clrb  fsk_tx_en         ; disable the FSK transmitter
            setb  fsk_rx_en         ; enable the FSK receiver
            call  @disable_o        ; disable the output

            retp

;*************************************************************
;
_FSK_RECEIVE                                ; FSK receiver starts here.
;
;*************************************************************
            bank  fsk_receive_bank
            add   fsk_trans_count,#1        ; Regardless of what is going on, increment the
            snc                     ; transition timer.  These get cleared when a transition
            jmp   :roll_over_error  ; takes place.
            cjb   fsk_trans_count,#low_high_th,:fsk_timer_out     ; as soon as it takes longer than 95
counts
```

```
            setb  fsk_current_in                              ; to transition, this must be a low
frequency
:fsk_timer_out
            mov   w,rb
            and   w,#%00000010            ; get the current state of rb.
            xor   w,rb_past_state         ; compare it with the previous state of the pin
            jz    fsk_rx_out          ; if there was no change, then jump out, there is nothing to do.
                                      ; Now it is time to determine if the transition that took place
indicates a bit was received
                                      ; (it must be within some thresholds... below 20, ignore it, below
40, what???,
                                      ; below 95, high frequency, below 140, low frequency (already
set), above 140,
                                      ; what???)
            cjb   fsk_trans_count,#glitch_th,:glitch_so_ignore          ; pulse was below specs,
ignore it... probably noise
            cjb   fsk_trans_count,#low_count_error_th,:error            ; pulse was not a glitch but
wasn't long enough to mean anything... huh?
            cjb   fsk_trans_count,#low_high_th,:high_frequency          ; pulse was within specs for a
high frequency...
            cjb   fsk_trans_count,#high_count_error_th,:fsk_receive_done      ; pulse was within specs
for a low frequency (don't do anything)
            jmp   :error                                          ; pulse was too long to mean
anything, so do nothing.
:high_frequency                           ; a high frequency corresponds to low data.
            clrb  fsk_current_in
            jmp   :fsk_receive_done

:roll_over_error                          ; if the counter rolls over, keep it in range.
;--------------- PUT ERROR HANDLING CODE IN HERE -----------------
            mov   fsk_trans_count,#high_count_error_th
            clr   fsk_rx_count
            jmp   :glitch_so_ignore
:error                                        ; if there is another type of error, just clear
                                          ; any UART receive.
;--------------- PUT ERROR HANDLING CODE IN HERE -----------------
            clr   fsk_rx_count

:fsk_receive_done
            clr   fsk_trans_count             ; clear the bit counter.
:glitch_so_ignore                         ; don't clear the counter if the data was a glitch
```

```
            mov   w,rb                    ; save the new state of RB.
            and   w,#%00000010
            mov   rb_past_state,w
fsk_rx_out
;*************************************************************************
:fsk_uart
; This is an asynchronous receiver.  Written by Craig Webb.  Modified by
; Chris Fogelklou for use with FSK receive routine.
;*************************************************************************
            bank  fsk_receive_bank
            jnb   divider.divider_bit,fsk_rx_done     ; (Divide operation frequency by divider_bit)
            movb    c,fsk_current_in                  ; get current rx bit
            test    fsk_rx_count                ; currently receiving byte?
            jnz     :rxbit                      ; if so, jump ahead
            mov     w,#9                        ; in case start, ready 9 bits
            sc                                  ; skip ahead if not start bit
            mov     fsk_rx_count,w                  ; it is, so renew bit count
            mov     fsk_rx_divide,#start_delay  ; ready 1.5 bit periods
:rxbit            djnz    fsk_rx_divide,fsk_rx_done         ; middle of next bit?
            setb    fsk_rx_divide.baud_bit      ; yes, ready 1 bit period
            dec     fsk_rx_count                    ; last bit?
            sz                                  ; if not
            rr      fsk_rx_byte                     ;  then save bit
            snz                                 ; if so
            setb    fsk_rx_flag                     ;  then set flag
fsk_rx_done
            RETP
;*************************************************************************
; END FSK ROUTINES
;*************************************************************************
;*************************************************************************
; DTMF generate lookup tables.  Gives the tone required for each of the
; DTMF digits.
;*************************************************************************
_0_         dw    f941_h,f941_l,f1336_h,f1336_l
_1_         dw    f697_h,f697_l,f1209_h,f1209_l
_2_         dw    f697_h,f697_l,f1336_h,f1336_l
_3_         dw    f697_h,f697_l,f1477_h,f1477_l
_4_         dw    f770_h,f770_l,f1209_h,f1209_l
_5_         dw    f770_h,f770_l,f1336_h,f1336_l
_6_         dw    f770_h,f770_l,f1477_h,f1477_l
```

```
_7_         dw    f852_h,f852_l,f1209_h,f1209_l
_8_         dw    f852_h,f852_l,f1336_h,f1336_l
_9_         dw    f852_h,f852_l,f1477_h,f1477_l
_star_            dw    f941_h,f941_l,f1209_h,f1209_l
_pound_           dw    f941_h,f941_l,f1477_h,f1477_l

;*************************************************************************
_ring_detect
;     Ring detect has 3 functions:
;           - It filters out the 20Hz pulsing of the ring-line by using
;             a software monostable with an 80ms timeout.  If the length
;             of time between pulses exceeds 80ms, the "ringing_1" flag gets
;             cleared.  This can also be used to show the pause between
;             rings in a distinctive ring pattern.
;           - It uses another monostable with a timeout of 530ms to
;             filter out distinctive ring pauses, whose largest pause
;             between distinctive rings is 525ms.  If the pause exceeds
;             the 530ms timeout, the "ring_occured" flag is set, indicating
;             that a ring is complete.
;           - A duration timer is used to find the amount of time the
;             line was ringing between pauses.  This can be used to
;             decode distinctive ring.  The register is called
;             ring_duration.  The duration is saved as soon as a pause is
;             found.
;*************************************************************************
      bank  ring_detect_bank

      ;*****************************************************************
      ; First run a timer that rolls over every 10ms, so all other timers
      ; can sync to it.
      ;*****************************************************************
      inc   ring_timer_low
      jnz   :no_roll_1
      inc   ring_timer_high
      jnz   :no_roll_1
      setb  ring_timer_flag
      mov   ring_timer_low,#$04
      mov   ring_timer_high,#$f4
:no_roll_1

      ;*****************************************************************
```

```
      ; Now create a monostable that times out after 80ms, because this
      ; is the maximum amount of time between two pulses of the ring
      ; signal.  Ringing gets cleared when this monostable times out.
      ;*****************************************************************

      jb    ring,:check_for_duration

:ring_low
      mov   ring_off_timer_1,#9                ; if the ring line is low, set up
      setb  ringing_1
      jb    ringing_2,:check_for_duration
      mov   ring_timer,#221
      setb  ringing_2

:check_for_duration
      jnb   ring_timer_flag,:ring_detect_done  ; jump out if the timer_flag is not set.
      clrb  ring_timer_flag

      jnb   ringing_1,:not_ringing_1
      inc   ring_duration_timer                ; count the ring duration
      snz
      setb  long_ring
      dec   ring_off_timer_1              ; while it is ringing.
      jnz   :not_ringing_1
:done_short_ring
      clrb  ringing_1                     ; clear the flag that indicates
                                          ; it is ringing.
      mov   ring_duration,ring_duration_timer   ; and save the duration
                                          ; of the ring.
      clr   ring_duration_timer                 ; reset the timer which times how
                                          ; long each ring is.
      setb  ring_pause                    ; indicate pause between rings.

:not_ringing_1                                  ; Now check if the 530ms monostable
                                          ; has timed out.
      jnb   ringing_2,:ring_detect_done
      dec   ring_timer
      jnz   :ring_detect_done
:done_whole_ring
      clrb  ringing_2
      setb  ring_captured
```

```
:ring_detect_done

      retp
;*************************************************************************
; Done ring detection interrupt service routine
;*************************************************************************
org   $600
;*************************************************************************
; DTMF transmit functions/subroutines
;*************************************************************************
;*************************************************************************
digit_2_index
; This subroutine converts a digit from 0-9 or a '*' or a '#' to a table
; lookup index which can be used by the load_frequencies subroutine.  To use
; this routine, pass it a value in the 'byte' register.  No invalid digits
; are used. (A, B, C, or D)
;*************************************************************************

            bank  serial
            cja   byte,#'9',:error        ; if the character is above 9, then error (get another char)
            cje   byte,#'*',:star
            cje   byte,#'#',:pound
            cjb   byte,#'0',:error
            sub   byte,#'0'               ; convert to decimal number
            jmp   :got_it
:star       mov   byte,#10
            jmp   :got_it
:pound            mov   byte,#11

:got_it           retp
:error
            mov   byte,#$0FF
            retp
;*************************************************************************
load_frequencies
; This subroutine loads the frequencies using a table lookup approach.
; The index into the table is passed in the byte register.
;*************************************************************************
            bank  serial
            cje   byte,#$0FF,:end_load_it
```

```
            clc
            rl    byte
            rl    byte                    ; multiply byte by 4 to get offset
            add   byte,#_0_               ; add in the offset of the first digit
            mov   temp,#4
            mov   fsr,#freq_count_high
            bank  serial

:dtmf_load_loop   mov   m,#5
            mov   w,byte
            IREAD                         ; get the value from the table
            bank  sin_gen_bank                  ; and load it into the frequency
            mov   indf,w                        ; register
            bank  serial
            inc   byte
            inc   fsr
            decsz temp
            jmp   :dtmf_load_loop               ; when all 4 values have been loaded,
:end_load_it      retp                          ; return
;*************************************************************************
dial_it           ; This subroutine puts out whatever frequencies were loaded
            ; for 60ms, and then stops outputting the frequencies.
;*************************************************************************
            bank  serial
            cje   byte,#$0FF,:end_dial_it
            bank  sin_gen_bank
            mov   curr_sin,#-4            ; use these values to start the wave at close to zero
crossing.
            mov   sinvel,#-8
            mov   curr_sin2,#-4           ; use these values to start the wave at close to zero
crossing.
            mov   sinvel2,#-8
            enable_o                ; enable the output
            mov   w,#6
            call  @delay_10n_ms           ; delay 20ms
            setb  dtmf_gen_en       ; dial the number
            mov   w,#11
            call  @delay_10n_ms           ; delay 100ms
            clrb  dtmf_gen_en       ; stop dialing
            call  @disable_o        ; now disable the outputs
:end_dial_it      retp
```

```
;*************************************************************************
sine_generator1                     ;(Part of interrupt service routine)
; This routine generates a synthetic sine wave with values ranging
; from -32 to 32.  Frequency is specified by the counter.  To set the
; frequency, put this value into the 16-bit freq_count register:
; freq_count = FREQUENCY * 6.83671552 (@50MHz)
;*************************************************************************
            bank  sin_gen_bank
            add   freq_acc_low,freq_count_low;2 ; advance sine at frequency
            jnc   :no_carry         ;2,4  ; if lower byte rolls over
            inc   freq_acc_high               ; carry over to upper byte
            jnz   :no_carry               ; if carry causes roll-over
            mov   freq_acc_high,freq_count_high ; then add freq counter to accumulator (which should
be zero,
                                      ; so move will work)
                                      ; and update sine wave
            jmp   :change_sin
:no_carry
            add   freq_acc_high,freq_count_high ; add the upper bytes of the accumulators
            jnc   :no_change
:change_sin

            mov   w,++sinvel  ;1          ; if the sine wave
            sb    curr_sin.7  ;1          ; is positive, decelerate
            mov   w,--sinvel  ;1          ; it.  Otherwise, accelerate it.
            mov   sinvel,w    ;1
            add   curr_sin,w  ;1          ; add the velocity to sin


:no_change

;*************************************************************************
sine_generator2                             ;(Part of interrupt service routine)
; This routine generates a synthetic sine wave with values ranging
; from -32 to 32.  Frequency is specified by the counter.  To set the
; frequency, put this value into the 16-bit freq_count register:
; freq_count = FREQUENCY * 6.83671552 (@50MHz)
;*************************************************************************

            bank  sin_gen_bank
            add   freq_acc_low2,freq_count_low2;2     ;advance sine at frequency
```

```
            jnc   :no_carry          ;2,4  ; if lower byte rolls over
            inc   freq_acc_high2                ; carry over to upper byte
            jnz   :no_carry                ; if carry causes roll-over
            mov   freq_acc_high2,freq_count_high2     ; then add freq counter to accumulator (which
should be zero,
                                           ; so move will work)
                                           ; and update sine wave
            jmp   :change_sin
:no_carry
            add   freq_acc_high2,freq_count_high2     ; add the upper bytes of the accumulators
            jnc   :no_change
:change_sin

            mov   w,++sinvel2 ;1           ; if the sine wave
            sb    curr_sin2.7 ;1           ; is positive, decelerate it
            mov   w,--sinvel2 ;1           ; it.  Otherwise, accelerate it.
            mov   sinvel2,w   ;1
            add   curr_sin2,w ;1           ; add the velocity to sin


:no_change
            jb    dtmf_gen_en,:do_DTMF
            mov   pwm0,curr_sin                ; mov the value of SIN into the PWM output
            add   pwm0,curr_sin2               ; mov the value of SIN2 into the PWM output
            clc
            rl    pwm0                     ; double the value of the PWM output
            add   pwm0,#128                ; put it in the middle of the output range
            retp                           ; return with page bits intact


:do_DTMF                                   ; If we are doing DTMF generation, then we need to add
"twist" to the
                                           ; signal (divide sin2 by 4 and add it to it's original

value)
            mov   pwm0,curr_sin2               ; mov sin2 into pwm0
            mov   IRQ_temp,w               ; mov the high_frequency sin wave's current value
            clc                             ; into a temporary register
            snb   IRQ_temp.7               ; divide temporary register by four by shifting right
            stc                             ; (for result = (0.25)(sin2))
            rr    IRQ_temp
            clc
```

```
                snb     IRQ_temp.7
                stc
                mov     w,>>IRQ_temp
                add     pwm0,w                          ; (1.25)(sin2) = sin2 + (0.25)(sin2)
                add     pwm0,curr_sin                   ; add the value of SIN into the PWM output
                                                ; for result = pwm0 = 1.25*sin2 + 1*sin
                add     pwm0,#128               ; put pwm0 in the middle of the output range (get rid of
negative values)
                retp                            ; return with page bits intact
```